High Level Architecture
Management Object Model

Version 0.2
October 17, 1996

1.

1. Introduction 3
2. HLA Management Object Model 4
3. HLA Management Interaction Model 6
4. Use of MOM 9
   4.1 Federate Object Updates 9
   4.2 Alerts 10
   4.3 Time Advance Control 11
5. Expansion of MOM Capabilities 12
   5.1 Increased State Data 12
   5.2 Added Query 14
6. Extension of the MOM Concept 16
   6.1 Federation Membership 16
   6.2 Ownership Passing 17
   6.3 Time Interruption 18
   6.4 State Saving 19
   6.5 Other Actions 21
7. References 22
8. Acronyms and Abbreviations 23

Introduction

This document presents a Federation Object Model (FOM) for the management functions of the DOD High Level Architecture's (HLA) Run Time Infrastructure (RTI). For brevity, it will be referred to as the Management Object Model or MOM.

The MOM uses the same mechanisms for the management of a federation as are used for the exchange of information among federates. So, for example, objects of class `Manager.Federate` contain attributes that describe the characteristics and state of a federate; interactions of class `Manager.Federate` permit queries and control of federates by a Manager federate.

The MOM differs from HLA data exchange mechanisms in several ways:

- MOM classes, attributes and parameters are pre-defined for every federation.
- The RTI is responsible for publishing, subscribing to, and generating many of the objects and interactions. Normally, no federate involvement is required.
- A manager federate need only subscribe to the `Manager` object class and subscribe to and publish appropriate `Manager` interaction class to have full access to understanding and control of the member federates of a federation.

The document is organized as follows:

Sections 2 and 3 present MOM information in a form consistent with the DMSO Object Model Template (OMT) described in [DMSO1]. Section 2 describes objects and their attributes; Section 3 describes interactions and their parameters. Understanding these sections requires a familiarity with the HLA time management philosophy presented in [DMSO2].

Section 4 describes the use of the mechanisms presented in Sections 2 and 3. This section assumes familiarity with the HLA data exchange mechanisms specified in [DMSO3].

Sections 5 and 6 propose extensions to the MOM beyond the functionality described above. Section 5 describes new capabilities and Section 6 proposes MOM implementations of functionality currently implemented through the RTI API.

Sections 7 and 8 contain references and acronym expansions.

2. HLA Management Object Model

This section presents the class structure for objects in the MOM and attributes associated with the class. The MOM has only one class, called `Manager` and three subclasses that describe aspects of a Federate, a Federation, and the RTI.

All updates of the `Manager` class use receive order, reliable services for data transmission. No attributes of class `Manager` are transferable.

Table 1. Object Class Structure

`Manager`	`Federate`
	`Federation`
	`RTI`

Table 2. Object Class Definition

Object Class	**Definition**
`Federate`	Federate and RTI functionality associated with it.
`Federation`	Federation and RTI functionality associated with it.
`RTI`	RTI characteristics

Table 3. Object Attributes

Class	**Attribute**	**Data Type**	**Update Type**	**Notes**
`Manager.` `Federate`	`FederateFederation`	text	static	
	`FederateHost`	text	static	
	`FederateHandle`	handle	static	
	`FederateLookahead`	float	periodic	[2]
	`FederateName`	text	static	
	`FederateState`	enumerated	conditional	[1]
	`FederateTime`	float	periodic	[2]
	`TimeConstrained`	boolean	conditional	[1]
	`TimeRegulating`	boolean	conditional	[1]
`Manager.` `Federation`	`FederationName`	text	static	
	`FederationState`	enumerated	conditional	[1]
	`FederationTime`	float	periodic	[3]
`Manager.RTI`	`RTIVersion`	text	static	

[note 1] Conditional attributes are updated only when the value changes.

[note 2] Periodic attributes to class `Manager.Federate` objects are updated at the rate set by the `Manager.Federation.Action.SetTiming` interaction.

[note 3] Periodic attributes to class `Manager.Federation` objects are updated at the rates set by the `Manager.Federation.Action.SetTiming` interaction.

Table 4. Object Attribute Definitions

Attribute	**Definition**
`FederateFederation`	The federation that a federate belongs to
`FederateHost`	The name of the computer that is hosting the federate.
`FederateHandle`	A value assigned to the federate.
`FederateLookahead`	The lookahead of the federate as determined by the federate.
`FederateName`	An arbitrary name. It need not be unique in the federation.
`FederateState`	The current activity of the federate. Possible values are `idle` awaiting directions `running` processing normally `saving` saving state `restoring` restoring state `damaged` incapable of processing normally `joining` joining the federation `resigning` resigning from the federation `resigned` resigned from the federation
`FederateTime`	The local time of the federate.
`FederationName`	The name of a federation
`FederationState`	The current activity of the federation. Possible values are `idle` awaiting directions `running` processing normally `saving` saving state `restoring` restoring state
`FederationTime`	The logical time of a federation
`RTIVersion`	The version of the RTI in use.
`TimeConstrained`	Whether the time advance of a federate is constrained by the time of other federates in the federation.
`TimeRegulating`	Whether a federate intends to regulate the time advance of other federates in the federation.

3. HLA Management Interaction Model

This section presents the interactions that are a part of the MOM. The interactions represent reports from individual federates on anomalies (`Manager.Federate.Alert`), queries from the manager federate to a federate (`Manager.Federate.Query`) or to a federation (`Manager.Federation.Query`), responses by federates to the queries (`Manager.Federate.Response`), and directives by the manager federate to a federate (`Manager.Federate.Action`) or to a federation (`Manager.Federation.Action`).

All interactions of the `Manager` class use receive order, reliable services for data transmission.

Table 5. Interaction Class Structure

`Manager`	`Federate`	`Action`	`SetTiming`
		`Alert`	
		`Response`	`WhoPublishes`
		`Response`	`WhoSubscribes`
	`Federation`	`Action`	`SetTiming`
		`Query`	`WhoPublishes`
		`Query`	`WhoSubscribes`

Note: the `Manager` interaction class structure is more complex than is needed for the capabilities presented—this permits future expansion of the MOM and extension by later federate developers.

Table 6. Interaction Definitions

Interaction Class	**Definition**
`Manager.Federate.Action.SetTiming`	Management federate specifies timing parameters for a federate.
`Manager.Federate.Alert`	Federate indicates that an anomaly has occurred.
`Manager.Federate.Response.WhoPublishes`	Federate replies to a publishing query if it is responsible for publishing an attribute of an object.
`Manager.Federate.Response.WhoSubscribes`	Federate replies to a subscribing query if it subscribes to an attribute of an object.
`Manager.Federation.Action.SetTiming`	Management federate specifies timing parameters for a federation
`Manager.Federation.Query.WhoPublishes`	Management federate asks which federate is publishing an attribute of an object.
`Manager.Federation.Query.WhoSubscribes`	Management federate asks which federate is subscribing to an attribute of an object.

Table 7. Interactions

Interaction	Initiating Class	Receiving Class	Parameters	Notes
Manager.Federate. Action.SetTiming	Federate	Federate	ToFederate Lookahead TimeConstrained TimeRegulating	
Manager.Federation. Action.SetTiming	Federate	Federate	ToFederation Federate-ReportFrequency Federation-ReportFrequency	
Manager.Federate. Alert	Federate		FromFederate AlertSeverity AlertText AlertID	
Manager.Federation. Query.WhoPublishes	Federate	Federation	ObjectID AttributeHandle	
Manager.Federation. Query.WhoSubscribes	Federate	Federation	ObjectID AttributeHandle	
Manager.Federate. Response.WhoPublishes	Federate	Federate	FromFederate ObjectID AttributeHandle	
Manager.Federate. Response. WhoSubscribes	Federate	Federate	FromFederate ObjectID AttributeHandle	

Table 8. Interaction Parameters

Parameter	Data Type	Units	Notes
AlertID	enumerated		
AlertSeverity	integer		
AlertText	text		
AttributeHandle	handle		
Federate-ReportFrequency	integer	Seconds	Elapsed time
Federation-ReportFrequency	integer	Seconds	Elapsed time
FromFederate	handle		
Lookahead	float		
ObjectID	handle		
TimeConstrained	boolean		
TimeRegulating	boolean		

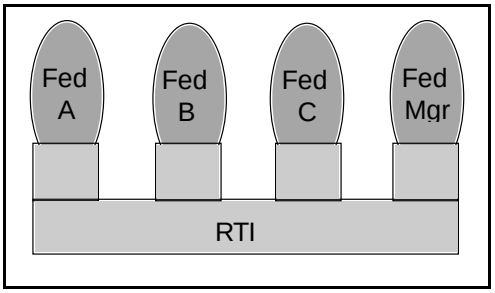
Parameter	Data Type	Units	Notes
ToFederate	handle		
ToFederation	handle		

Table 9. Parameter Definitions

Parameter	Definition
AlertID	An identifier associated with an alert.
AlertSeverity	Severity of an alert.
AlertText	Text describing the alert.
AttributeHandle	The handle of an attribute.
Federate-ReportFrequency	The wall clock time interval between updates of the Federate-Time and FederateLookahead attributes of objects of class Manager.Federate belonging to a federation
Federation-ReportFrequency	The wall clock time interval between updates of the Federa-tionTime attribute of an object of class Manager.Federation.
FromFederate	Federate initiating the interaction
Lookahead	The logical lookahead value of the federate.
ObjectID	The identifier of an object
TimeConstrained	Whether the time advance of a federate should be constrained by the time of other federates.
TimeRegulating	Whether a federate should participate in the time advance of other federates.
ToFederate	Federate targeted for the interaction.
ToFederation	Federation targeted for the interaction.

4. Use of MOM

This section discusses the use of the MOM in a federation. The functioning of the elements of the MOM is presented in the form of examples. Since the MOM is primarily intended for use by a manager federate, all examples contain one. All examples also use the following figure to illustrate activity—the figure depicts a federation of three normal federates and one Manager federate.



By default, the publishing, subscribing, and generation status of classes, objects, and interactions is as follows:

- The RTI publishes (but does not subscribe to) the `Manager.Federate` class for each federate, the `Manager.Federation` class for each federation and the `Manager.RTI` class.
- The RTI invokes the **Update Attribute Values** service for defined attributes of the `Manager.Federate` object class of each federate, the `Manager.Federation` object class for each federation and the `Manager.RTI` object class as appropriate.
- The RTI publishes (but does not subscribe to) the `Manager.Federate.Alert` and `Manager.Federate.Response` interaction classes for each federate.
- The RTI subscribes to (but does not publish) the `Manager.Federate.Action`, the `Manager.Federation.Action`, and `Manager.Federation.Query` interaction classes for each federate and federation.
- The RTI generates interactions of class `Manager.Federate.Alert` and `Manager.Federate.Response` for each federate as appropriate.

Normally, a Manager federate publishes, subscribes to, and generates classes, objects, and interactions is as follows:

- The Manager federate subscribes to and publishes all desired subclasses of the `Manager` object class.
- The Manager federate subscribes to all desired subclasses of the `Manager.Federate.Alert` and `Manager.Federate.Response` interaction classes.
- The Manager federate publishes and generates interactions of all subclasses of the `Manager.Federate.Action`, `Manager.Federation.Action` and `Manager.Federate.Query` interaction classes.

4.1 Federate Object Updates

Assume that the federation is executing under the following conditions:

- Federates A, B, and C have not received any directions regarding timing.
- The RTI automatically publishes all specified attributes for the `Manager.Federate` object class for each federate and registers an instance of the `Manager.Federate` object class for each federate.

· The Manager federate subscribes to all desired attributes of the `Manager.Federate` object class.

4.1.1 Snapshot

If the Manager federate is interested in a snapshot of the status of all federates in a federation, the following sequence of information exchanges occurs

1. *Fed Mgr* invokes the **Request Class Attribute Value Update** service for the `Manager.Federate` class.
2. The RTI invokes the **Update Attribute Values** service for each federate. Since the RTI can respond to all aspects of the request, it does not pass any aspect of the request to the federates.
3. The RTI sends the information to the *Fed Mgr* by invoking the **Reflect Attribute Values** service. When all information from all federates is received, the Manager federate has its snapshot data.

4.1.2 Regular Updates

If the Manager federate is interested in receiving regular updates of the status of all federates in a federation, the following sequence of information exchanges occurs

1. *Fed Mgr* sends an interaction of class `Manager.Federation.Action.SetTiming` with the desired interval provided in parameter `FederateReportFrequency`.
2. The RTI sets internal timers appropriate to the `FederateReportFrequency` parameter. Since the RTI can respond to all aspects of the interaction, it does not pass any aspect of the request to the federates.
3. When a timer expires for a federate, the RTI invokes the **Update Attribute Values** service for the appropriate `Manager.Federate` object and resets the timer to the specified interval.
4. The RTI sends the information to *Fed Mgr* by invoking the **Reflect Attribute Values** service.

4.2 Alerts

Assume that the federation is executing under the following conditions:

· The RTI automatically publishes interactions for the `Manager.Federate.alert` interaction classes for each federate.

· The Manager federate subscribes to interactions of the `Manager.Federate.alert` class.

4.2.1 Federate Anomaly

If some anomaly occurs in federate A,

1. *Fed A* invokes the **Send Interaction** service with class `Manager.Federate.Alert` and information describing the anomaly.
2. The RTI ignores the interaction for *Fed B* and *Fed C* since neither had subscribed to interactions of class `Manager.Federate.Alert`.
3. The RTI passes the information to *Fed Mgr* by invoking the **Receive Interaction** service; *Fed Mgr* processes the alert accordingly.

4.2.2 RTI Anomaly

If some anomaly occurs in the RTI functionality,

1. The RTI invokes the **Send Interaction** service with class `Manager.Federate.Alert` and information describing the anomaly.
2. The RTI does not send the interaction to *Fed A*, *Fed B*, and *Fed C* since they had not subscribed to interactions of class `Manager.Federate.Alert`.
3. The RTI passes the information to *Fed Mgr* by invoking the **Receive Interaction** service; *Fed Mgr* processes the alert accordingly.

4.3 Time Advance Control

The Manager federate can control the extent to which a federate uses the logical times of other federates to control its own time advance and to participate in other federates logical time advance.

4.3.1 Time Constrained

If federates A, B, and C are fully participating in logical time advance, but it is desired to remove the constraint of coordinated time advance for federate C,

1. *Fed Mgr* invokes the **Send Interaction** service with class `Manager.Federate.Action.SetTiming` and parameter `TimeConstrained` set to `False` addressing *Fed C.*
2. The RTI subsequently permits *Fed C* to advance its logical time without regard to the logical times of *Fed A* and *Fed B*. *Fed C* is not aware of the action. The RTI invokes the **Update Attribute Values** service for *Fed C* with attribute `TimeConstrained` set to `false`.

4.3.2 Time Regulating

If federates A, B, and C are fully participating in logical time advance, but it is desired that federate C should no longer participate in the logical time advance for federates A and B,

1. *Fed Mgr* invokes the **Send Interaction** service with class `Manager.Federate.Action.SetTiming` and parameter `TimeRegulating` set to `False` addressing *Fed C.*
2. The RTI removes reference to *Fed C* from consideration for time advance for *Fed A* and *Fed B.* All subsequent output from *Fed C* is treated as if it were using Receive Order services. The RTI invokes the **Update Attribute Values** service for *Fed C* with attribute `TimeRegulating` set to `false`.

4.3.3 Lookahead

If federates A, B, and C are fully participating in logical time advance, but it is desired to change the lookahead value for federate C,

1. *Fed Mgr* invokes the **Send Interaction** service with class `Manager.Federate.Action.SetTiming` and parameter `Lookahead` set to the a new value addressing *Fed C.*
2. The RTI changes the internal value for lookahead for *Fed C* and uses this value for new data emanating from *Fed C*. The RTI invokes the **Update Attribute Values** service for *Fed C* with attribute `FederateLookahead` set to the new value.

5. Expansion of MOM Capabilities

The MOM functionality described in Sections 2, 3, and 4 of this document depict a minimal capability necessary for implementing a federate that can effectively manage a federation. This section proposes expansion of that functionality to permit a more robust Manager federate.

5.1 Increased State Data

Attributes could be added to the `Manager` object class to provide a broader insight into the status and health of the federate. The expansion proposed here breaks out the number of objects that are owned and reflected by the federate, describes the amount of activity passing to and from the federate, and provides insight into the queues that the RTI maintains for each federate.

These added attributes would be used and accessed in the same manner as those described in Section 4.1. Added timer parameters are specified below to permit control of the frequency of update of each set of state data.

In FOM terms, the Object Attributes table is expanded as follows:

Class	Attribute	Data Type	Update Type	Notes
`Manager.`	`ClassesOwned`	vector	periodic	[1]
`Federate`	`QueueFIFOLength`	integer	periodic	[2]
	`QueueTSOHead`	float	periodic	[2]
	`QueueTSOLength`	integer	periodic	[2]
	`InDiscovers`	integer	periodic	[3]
	`InInteractions`	integer	periodic	[3]
	`InRemoves`	integer	periodic	[3]
	`InRetractions`	integer	periodic	[3]
	`InUpdates`	integer	periodic	[3]
	`OutDeletes`	integer	periodic	[3]
	`OutInteractions`	integer	periodic	[3]
	`OutRetractions`	integer	periodic	[3]
	`OutUpdates`	integer	periodic	[3]

[note 1] Attribute `ClassesOwned` is updated at the frequency set by the `OwnedReportFrequency` parameter to the `Manager.Federation.Action.SetTiming` interaction.

[note 2] Attributes `QueueFIFOLength` and `QueueTSO` are updated at the frequency set by the `QueueReportFrequency` parameter to the `Manager.Federation.Action.SetTiming` interaction.

[note 3] Attributes `In` ... and `Out` ... are updated at the frequency set by the `TransportReportFrequency` parameter to the `Manager.Federation.Action.SetTiming` interaction.

The Object Attribute Definitions table is expanded as follows:

Attribute	Definition
`ClassesOwned`	Vector of data triplets; each triplet describes the number of objects owned and reflected by this federate and consists of `class` an object class `owned` the number of objects of `class` whose `PrivilegeToDelete` attribute is owned by the federate `reflected` the number of objects of `class` whose `PrivilegeToDelete` attribute is reflected by the federate
`InDiscovers`	Number of **Discover Object** services processed by the federate
`InInteractions`	Number of **Receive Interaction** services processed by the federate
`InRemoves`	Number of **Remove Object** services processed by the federate
`InRetractions`	Number of **Reflect Retraction** services processed by the federate
`InUpdates`	Number of **Reflect Attribute Value** services processed by the federate
`OutDeletes`	Number of **Delete Object** services invoked by the federate
`OutInteractions`	Number of **Send Interaction** services invoked by the federate
`OutRetractions`	Number of **Retract** services invoked by the federate
`OutUpdates`	Number of **Update Attribute Value** services invoked by the federate
`QueueFIFOLength`	The number of entries in the FIFO queue maintained for the federate
`QueueTSOHead`	The time on the message that is at the head of the TSO `queue.`
`QueueTSOLength`	The number of entries in the TSO queue.

The Interactions table is expanded to add three parameters to the `Federate.Action.SetTiming` interaction as follows:

Interaction	Initiating Class	Receiving Class	Parameters	Notes

`Manager.Federation.Action.SetTiming`	Federate	Federate	`TransportReportFrequency`
			`OwnedReportFrequency`
			`QueueReportFrequency`	

The Interactions Parameters table is expanded as follows:

Parameter	Data Type	Units	Notes
Owned-ReportFrequency	integer	Seconds	Elapsed time
Queue-ReportFrequency	integer	Seconds	Elapsed time
Transport-ReportFrequency	integer	Seconds	Elapsed time

The Parameters Definitions table is expanded as follows:

Parameter	Definition
Owned-ReportFrequency	The wall clock time interval between federate updates of the ClassesOwned attribute of the Manager.Federate objects in a federation
Queue-ReportFrequency	The wall clock time interval between federate updates of the QueueFIFOLength, QueueTSOHead, and QueueTSOLength attributes of the Manager.Federate objects in a federation
Transport-ReportFrequency	The wall clock time interval between federate updates of the attributes beginning In... and Out... in the Manager.Federate object in a federation

5.2 Added Query

Queries can be added to the Manager interaction class to permit deeper probing into the state of the federate. Queries are used instead of object attributes because of the likely infrequency of request and the volume of information requested. The two queries proposed here ask for which objects are owned and reflected by a federate.

In FOM terms, the Interaction Class Structure table is expanded as follows:

Manager	Federate	Query	OwnsWhat
			ReflectsWhat
		Response	OwnsWhat
			ReflectsWhat

The Interactions table is expanded to add three parameters to the Federate.Action.Set-Timing interaction as follows:

Interaction	Initiating Class	Receiving Class	Parameters	Notes
Manager.Federate.Query.OwnsWhat	Federate	Federate	ToFederate	
Manager.Federate.Query.ReflectsWhat	Federate	Federate	ToFederate	
Manager.Federate.Response.OwnsWhat	Federate	Federate	FromFederate ObjectsOwned	
Manager.Federate.Response.ReflectsWhat	Federate	Federate	FromFederate ObjectsReflected	

The Interactions Parameters table is expanded as follows:

Parameter	Data Type	Units	Notes
`ObjectsOwned`	vector		
`ObjectsReflected`	vector		
`ToFederate`	handle		

The Parameters Definitions table is expanded as follows:

Parameter	Definition
`ObjectsOwned`	Vector of data pairs; each pair describes an object whose `PrivilegeToDelete` attribute is owned by the federate `Class` The class of the object `ObjectID` The identifier of the object.
`ObjectsReflected`	Vector of data pairs; each pair describes an object whose `PrivilegeToDelete` attribute is reflected by the federate `Class` The class of the object `ObjectID` The identifier of the object.
`ToFederate`	Federate targeted for the interaction.

6. Extension of the MOM Concept

This section proposes that the MOM concept could be extended to replace the API in many of the control functions performed by the RTI. The paragraphs below describe how Federate class interactions could be used to perform the API activities.

Advantages to this approach include:

- **Extensible**. The definitions included in this document are fixed for all RTI uses (all Federations know about these objects and interactions whether they use them or not). However, they can be extended in two ways: (1) by increasing the capabilities of the RTI, and (2) by increasing the capabilities of the federates. Since the objects and interactions follow all HLA rules, they can be extended by the federates and used as needed.

- **Loggable**. To the extent that any information passed by the RTI is loggable, the MOM object and interaction activities are loggable. No special logic is needed to capture the management activity as is needed with the API approach.

- **Intuitive**. The MOM approach uses the HLA methodology and documentation standards. As such, it should be familiar and intuitive to developers and users of federates. It simplifies toward a single approach to control of the federation.

Disadvantages to this approach include:

- **Documentation Changes**. The current documentation of the RTI [DMSO3] describes an API approach to control, both from the viewpoint of a normal federate and from that of a manager federate. This documentation would need to be revised. Similarly, the documentation for the MOM would need to become part of the HLA document set.

- **Queue Control**. The RTI must exercise additional control over queues for each federate. It must treat activity with class `Manager` in a special manner (attribute updates, interactions, etc.): this activity must go to the head of the queues so that it is treated internally as it arrives and is processed ahead of all other calls to the Federate.

6.1 Federation Membership

A Manager federate could control the membership in a federation using interactions.

If a manager federate wanted to cause a federate to resign from a federation, the following sequence of information exchanges might occur

1. *Fed Mgr* invokes the **Send Interaction** service for *Fed C* with class `Manager.Federate.Action.Resign`.
2. The RTI causes *Fed C* to resign from the federation. Since the RTI can respond to all aspects of the interaction, it does not pass any aspect of the request to the federate.
3. During the resigning process, the RTI invokes the **Update Attribute Values** service for *Fed C* with attribute `FederateState` set to `resigning`. After completion of the resigning process, the RTI invokes the **Update Attribute Values** service for *Fed C* with attribute `FederateState` set to `resigned`.

In FOM terms, the Interaction Class Structure table is expanded as follows:

Manager	Federate	Action	Resign

The Interactions Definitions table is expanded as follows:

Interaction Class	Definition
Manager.Federate.Action.Resign	Management federate directs another federate to resign from a federation.

The Interactions table is expanded as follows:

Interaction	Initiating Class	Receiving Class	Parameters	Notes
Manager.Federate.Action.Resign	Federate	Federate	ToFederate	

The Interactions Parameters table is expanded as follows:

Parameter	Data Type	Units	Notes
ToFederate	handle		

The Parameter Definitions table is expanded as follows:

Parameter	Definition
ToFederate	Federate targeted for the resign.

6.2 Ownership Passing

A Manager federate could control the ownership of attributes of objects.

If the manager federate wanted to cause a federate to take ownership of a set of attributes belonging to an object, the following sequence of information exchanges might occur

1. *Fed Mgr* invokes the **Send Interaction** service with class Manager.Federate.Action.TakeOwnership destined for *Fed C*, an object handle in parameter Object, and a set of attributes in parameter Attributes.
2. The RTI passes the direction to *Fed C* and ownership is assumed using existing API mechanisms (using the **Request Attribute Ownership Acquisition** service).

If the manager federate wanted to cause a federate to give up ownership of a set of attributes belonging to an object, the following sequence of information exchanges might occur

1. *Fed Mgr* invokes the **Send Interaction** service with class Manager.Federate.Action.ReleaseOwnership destined for *Fed C*, an object handle in parameter Federation, and a set of attributes in parameter Attributes.
2. The RTI passes the direction to *Fed C* and ownership is assumed using existing API mechanisms (using the **Request Attribute Ownership Divestiture** service).

In FOM terms, the Interaction Class Structure table is expanded as follows:

Manager	Federate	Action	TakeOwnership
			ReleaseOwnership

The Interactions Definitions table is expanded as follows:

Interaction Class	Definition
`Manager.Federate.Action.TakeOwnership`	Management federate directs another federate to take ownership of attributes of an object.
`Manager.Federate.Action.ReleaseOwnership`	Management federate directs another federate to release ownership of attributes of an object.

The Interactions table is expanded as follows:

Interaction	Initiating Class	Receiving Class	Parameters	Notes

`Manager.Federate.Action.TakeOwnership`	Federate	Federate	`ToFederate`
			`Object`
			`Attributes`	

`Manager.Federate.Action.ReleaseOwnership`	Federate	Federate	`ToFederate`
			`Object`
			`Attributes`	

The Interactions Parameters table is expanded as follows:

Parameter	Data Type	Units	Notes
`Attributes`	vector		
`Object`	handle		
`ToFederate`	handle		

The Parameter Definitions table is expanded as follows:

Parameter	Definition
`Attributes`	A vector of attribute handles.
`Object`	A specific object handle.
`ToFederate`	Federate targeted for the interaction.

6.3 Time Interruption

A Manager federate could stop and start the evolution of time in the federation.

If the manager federate wanted to pause the federation time advance (to break for lunch, for example), the following sequence of information exchanges might occur

1. *Fed Mgr* invokes a **Send Interaction** service with class `Manager.Federation.Action.Pause` and a future time in parameter `Time`.
2. When the time is reached, the RTI ceases passing TSO or FIFO information to all federates and passes pause order to all federates; and the federates pause using existing API mechanisms (using the **Initiate Pause** service).
3. The RTI invokes the **Update Attribute Values** service for all federates with attribute `FederateState` set to `idle`.

If the manager federate wanted to resume the federation time advance (after the break for lunch, for example), the following sequence of information exchanges might occur

1. *Fed Mgr* invokes the **Send Interaction** service with class `Manager.Federation.Action.Resume`.
2. The RTI resumes passing TSO or FIFO information to all federates and passes the direction to all federates; and the federates resume from the pause using existing API mechanisms (using the **Initiate Resume** service).
3. The RTI invokes the **Update Attribute Values** service for all federates with attribute `FederateState` set to `running`.

In FOM terms, the Interaction Class Structure table is expanded as follows:

`Manager`	`Federation`	`Action`	`Pause`
			`Start`

The Interactions Definitions table is expanded as follows:

Interaction Class	Definition
`Manager.Federation.Action.Pause`	Manager federate directs other federates to pause.
`Manager.Federation.Action.Start`	Manager federate directs other federates to resume running.

The Interactions table is expanded as follows:

Interaction	Initiating Class	Receiving Class	Parameters	Notes
`Manager.Federation.Action.Pause`	Federate	Federate	`Federation` `Time`	
`Manager.Federation.Action.Start`	Federate	Federate	`Federation`	

The Interactions Parameters table is expanded as follows:

Parameter	Data Type	Units	Notes
`Federation`	handle		
`Time`	float		

The Parameter Definitions table is expanded as follows:

Parameter	Definition
`Federation`	The federation that is to take action on the order
`Time`	The federate time that an activity (pause, save, restore) should occur.

6.4 State Saving

A Manager federate could cause state to be saved and restored for the federation.

If the federation manager wanted to cause the federation to save its state at federation time 12.000 and give the saved state an identifier of “george”, the following sequence of information exchanges might occur

1. *Fed Mgr* invokes the **Send Interaction** service with class `Manager.Federation.Action.Save`, 12.000 in parameter `Time` and “`george`” in parameter `SaveID`.
2. When the time is reached, the RTI ceases passing TSO or FIFO information to all federates and passes the save order to all federates and begins to save its own state; the federates save state using existing API mechanisms (using the **Initiate Federate Save** service).
3. During the saving process, the RTI invokes the **Update Attribute Values** service for each federate with attribute `FederateState` set to `saving`.
4. After completion of the saving process, the RTI invokes the **Update Attribute Values** service for each federate with attribute `FederateState` set to `running; it then` resumes passing TSO and FIFO information.

If the federation manager wanted to cause the federation to restore its state from a saved state with an identifier of “george”, the following sequence of information exchanges might occur

1. *Fed Mgr* invokes the **Send Interaction** service with class `Manager.Federation.Action.Restore` and “`george`” in parameter `SaveID`.
2. The RTI ceases passing TSO or FIFO information to all federates, passes the restore order to all federates, and begins restoring the RTI state; the federates restore state using existing API mechanisms (using the **Initiate Restore** service).
3. During the restoring process, the RTI invokes the **Update Attribute Values** service for each federate with attribute `FederateState` set to `restoring`.
4. After completion of the restoring process, the RTI invokes the **Update Attribute Values** service for each federate with attribute `FederateState` set to `idle`.

In FOM terms, the Interaction Class Structure table is expanded as follows:

`Manager`	`Federation`	`Action`	`Restore`
			`Save`

The Interactions Definitions table is expanded as follows:

Interaction Class	Definition
`Manager.Federation.Action.Restore`	Management federate directs other federates to restore their state.
`Manager.Federation.Action.Save`	Management federate directs other federates to save their state.

The Interactions table is expanded as follows:

Interaction	Initiating Class	Receiving Class	Parameters	Notes
`Manager.Federation.Action.Restore`	Federate	Federate	`Federation` `SaveID`	
`Manager.Federation.`	Federate	Federate	`Federation`	

Interaction	Initiating Class	Receiving Class	Parameters	Notes
Action.Save			Time SaveID	

The Interactions Parameters table is expanded as follows:

Parameter	Data Type	Units	Notes
Federation	handle		
SaveID	text		
Time	float		

The Parameter Definitions table is expanded as follows:

Parameter	Definition
Federation	The federation that is to perform the save or restore.
SaveID	Identifier to associate with a state save.
Time	The federate time that a save should occur.

6.5 Other Actions

Other actions that could be performed using the Manager class of objects and attributes include

- Changing the values of attributes
- Reporting the classes of objects and interactions that were subscribed to by a federate and changing the subscriptions using a Manager interaction (subscribing and unsubscribing).
- Reporting the classes of objects and interactions that were published to by a federate and changing the publications using a Manager interaction (publishing and unpublishing).

7. References

[DMSO1] *DOD High Level Architecture Object Model Template*, version 1.0, Defense Modeling and Simulation Office, 15 August 1996

[DMSO2] *HLA Time Management: Design Document*, version 1.0, Defense Modeling and Simulation Office, 15 August 1996

[DMSO3] *HLA Interface Specification*, version 1.0, Defense Modeling and Simulation Office, 15 August 1996

8. Acronyms and Abbreviations

API	Application Program Interface
DMSO	Defense Modeling and Simulation Office
DOD	Department of Defense
FIFO	First-In, First-Out
FOM	Federation Object Model
HLA	High Level Architecture
MOM	Management Object Model
OMT	Object Model Template
RO	Receive Order
RTI	Run Time Infrastructure
TSO	Time Stamp Order